Distribuované databázy

- Motivácia
- Homogénne a heterogénne databázové systémy
- Distribuované databázové systémy a transakcie
  - Požiadavky na systém, architektúra
  - Algoritmy

Literatúra

P.A. Bernstein, V. Hadzilacos, N. Goodman: Concurrency Control and Recovery in Database Systems

H. Garcia-Molina, J.D. Ullman, J. Widom: Database System Implementation, Prentice Hall, 2000

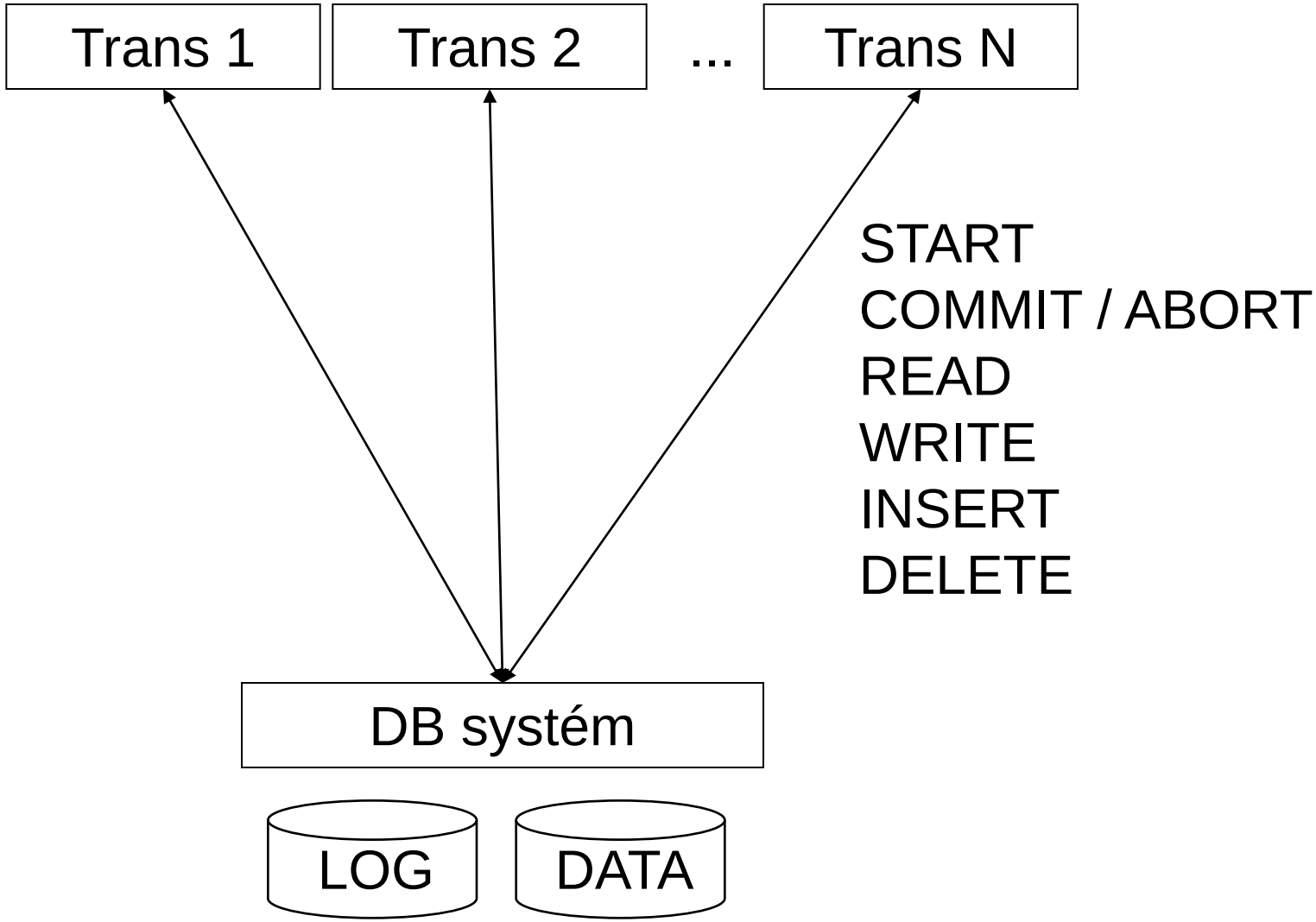
Centralizovaná (2-tier) architektúra

Centralizovaná databáza

- Dáta sú na jednom mieste
- Centrálna správa dát a transakcií
- Architektúra klient-server (2-tier)
  - Klient posiela transakčné operácie
  - Server spracúva prichádzajúce operácie, stará sa o splnenie ACID podmienok, atď.

Takto sa to dnes robí... Asi aj preto, že iné vlastne nie je

Centralizovaná (2-tier) architektúra



Centralizovaná (2-tier) architektúra

Výhody centralizovanej (2-tier) architektúry:

- jednoduchosť, "spoľahlivosť" (v zmysle odchytania programátorských chýb)
- kompatibilita s existujúcou byrokraciou

Nevýhody centralizovanej (2-tier) architektúry:

- všetko ostatné (nespoľahlivosť—ak vypadne centrála, dáta nie sú dostupné a treba počkať na obnovu; neškálovateľnosť—ak rýchlosť servera nestačí pre obslúženie spracovania požiadaviek, tak treba počkať na rýchlejší hardware; atď.)

Distribuované databázy

- Homogénne systémy
  - – Rovnaký software, rovnaká databázová schéma, rôzne dáta v rôznych uzloch
  - – **Cieľ: poskytnúť každému uzlu centralizovaný pohľad na dáta**
- Heterogénne systémy
  - – Rôzny software, rôzne databázové schémy v rôznych uzloch
  - – **Cieľ: integrovať existujúce databázové systémy do funkčného celku** (napr. rezervácia letenky a zároveň hotela cez WWW u dvoch rôznych spriatelených firiem)

Distribuované databázy

Je prirodzené pracovať s distribuovanými dátami:

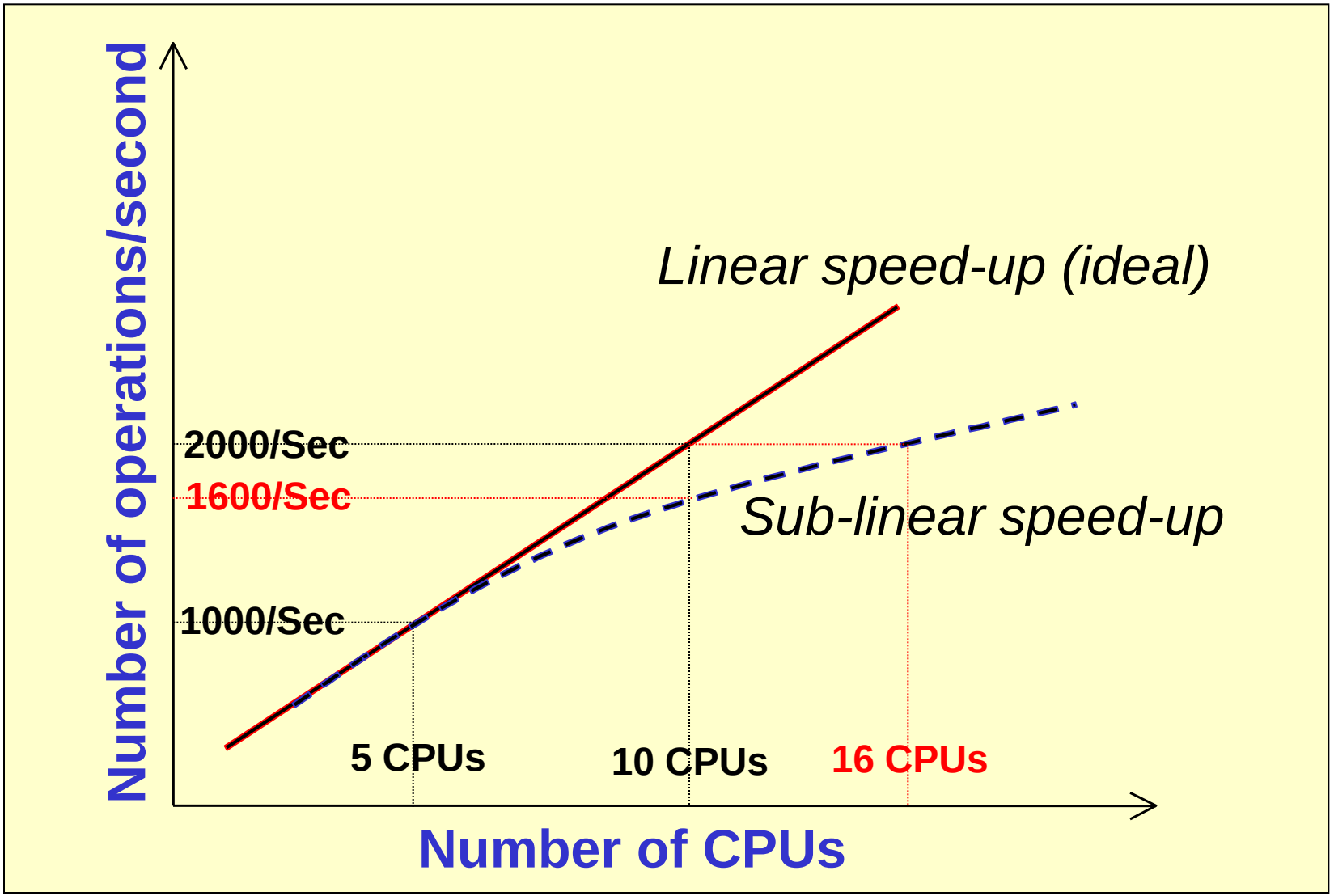
• Rôzne štátne inštitúcie pracujú s rôznymi údajmi o tých istých osobách: **vertikálna fragmentácia dát**

• Firma má veľa filiálok: **horizontálna fragmentácia dát**

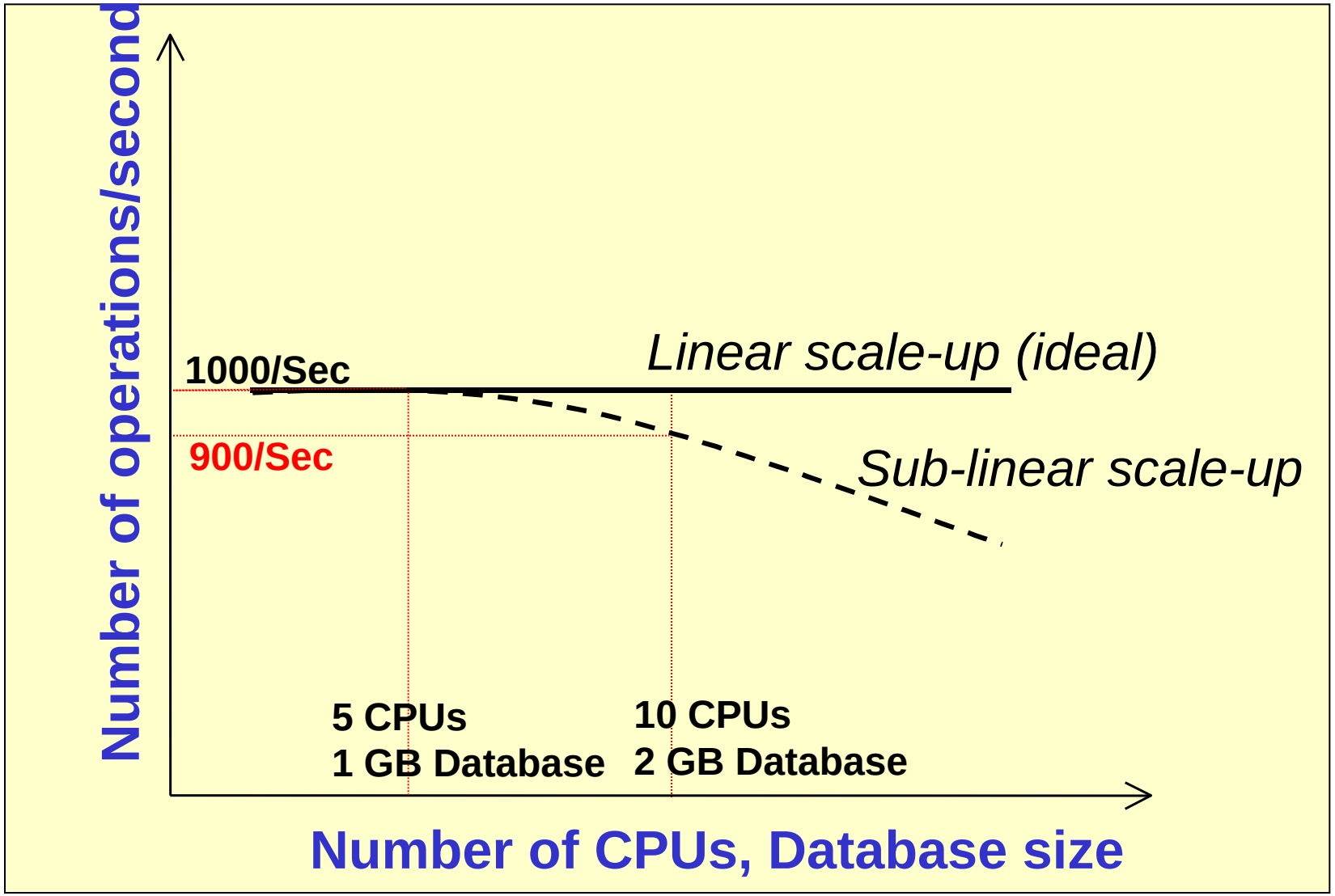
Budeme sa zaoberať horizontálnou fragmentáciou. Príklad: banková pobočka má „u seba“ len účty svojich klientov. Celá banka je zjednotením filiálok

Ak klient príde do ktorejkoľvek pobočky, tak je vo svojej banke a chce vedieť narábať so svojimi účtami—tak, ako keby bol vo svojej “domácej pobočke”

SPEEDUP


Number of operations/second
Linear speed-up (ideal)
2000/Sec
1600/Sec
Sub-linear speed-up
1000/Sec
5 CPUs
10 CPUs
16 CPUs
Number of CPUs

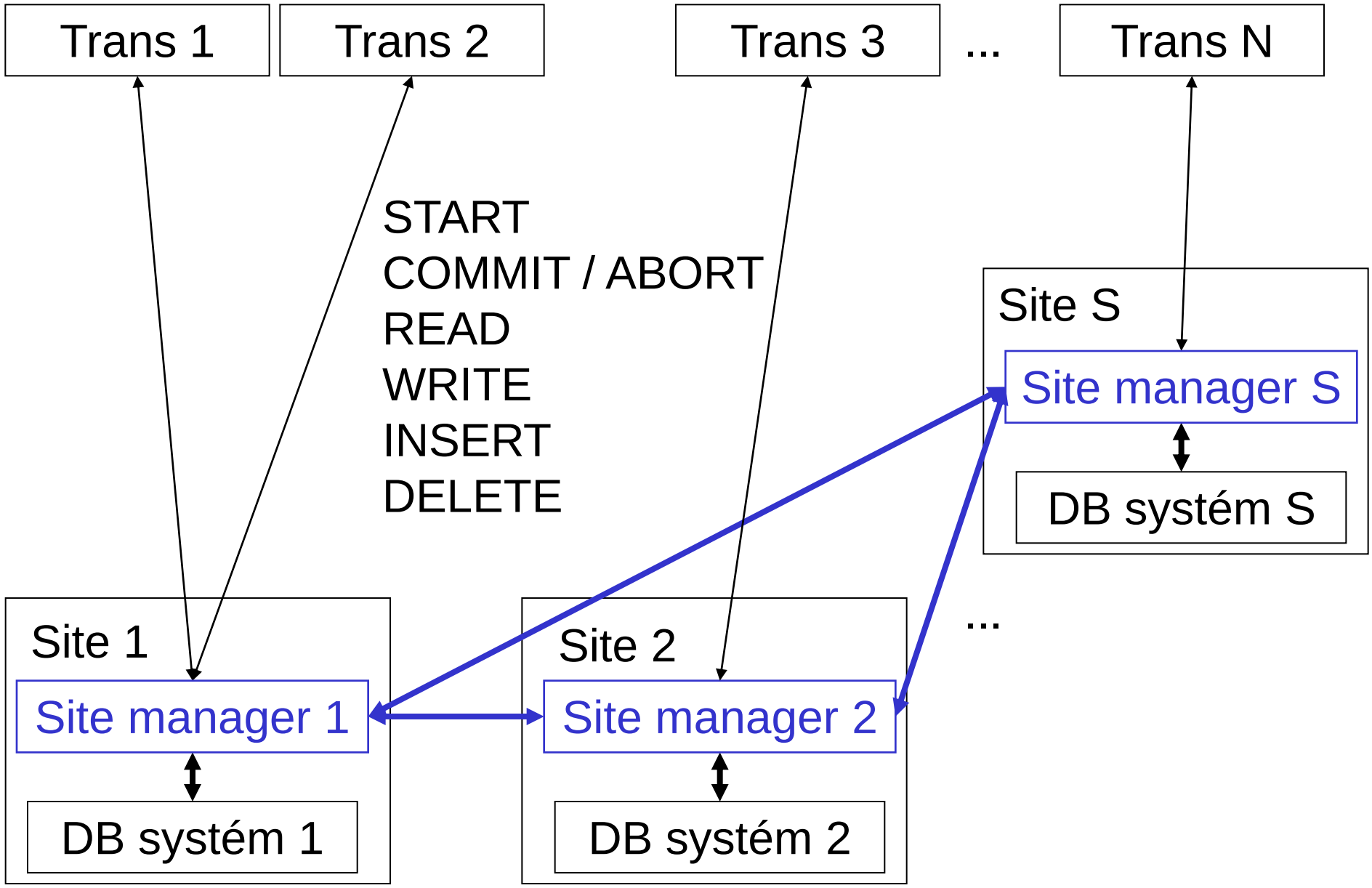
SCALEUP


Number of operations/second
1000/Sec
900/Sec
Linear scale-up (ideal)
Sub-linear scale-up
5 CPUs
1 GB Database
10 CPUs
2 GB Database
Number of CPUs, Database size

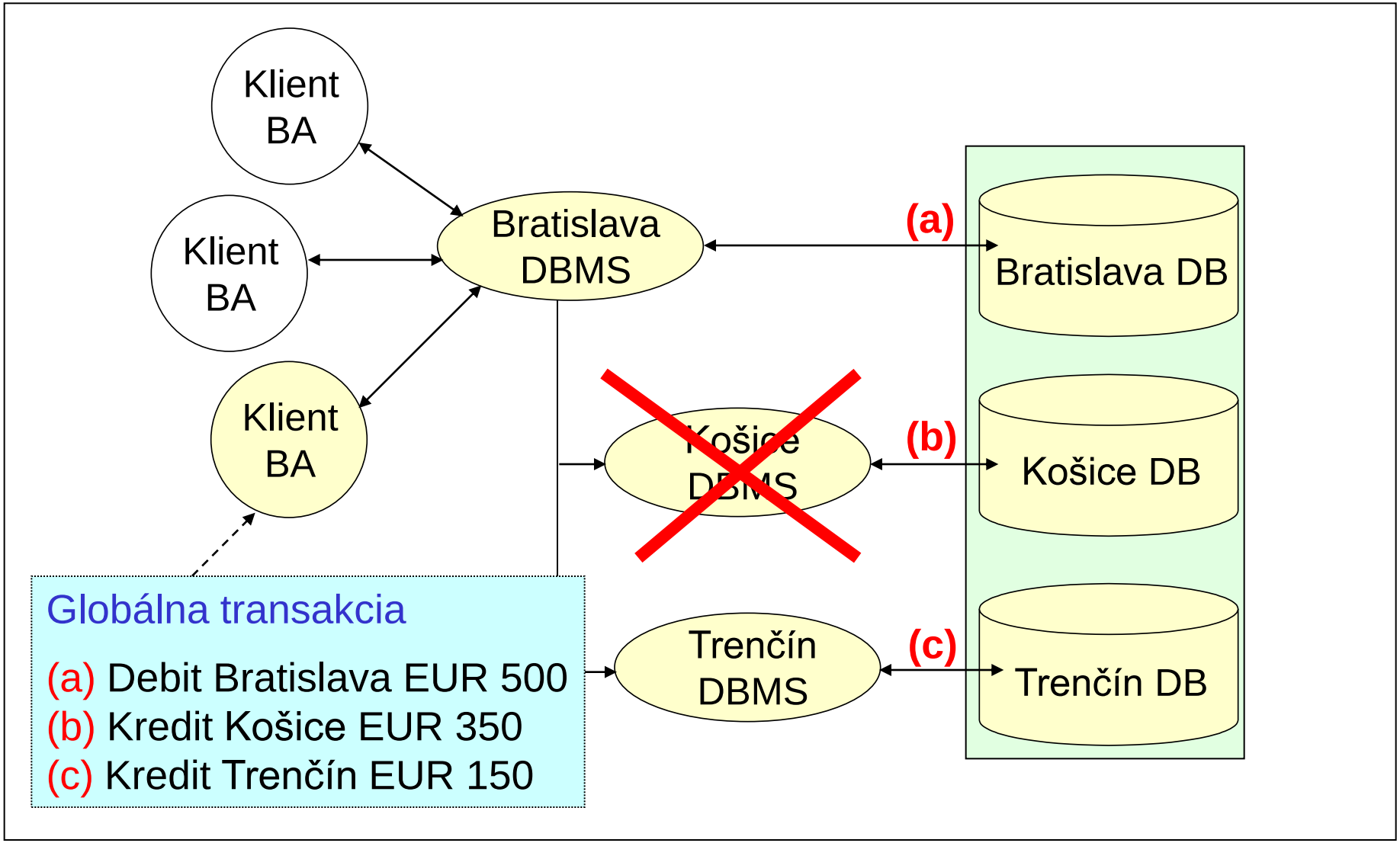
Distribuované databázy: pravidlá hry

- Požiadavky na homogénny databázový systém
  - ACID
  - **Transparentnosť distribuovanosti pre aplikácie (transakcie)**
- Dodatočné požiadavky
  - Paralelizmus
  - Odolnosť voči výpadkom uzlov a komunikačných liniek: **žiaden uzol nesmie nekonečne dlho čakať na obnovenie iného spadnutého uzlu**
  - Nízka réžia

Distribuovaná (3-tier) architektúra



Distribuované transakcie: nové problémy



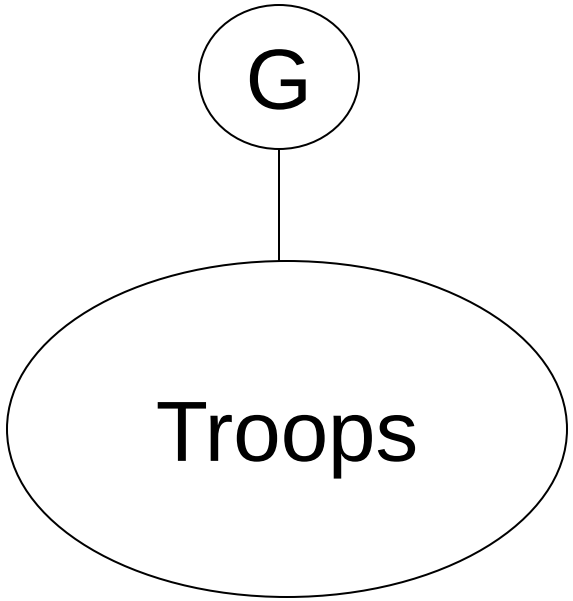
Nestačí splniť ACID v každom DBMS individuálne!

Atomicita: atomický commit protokol

Atomický commit protokol, ciele:

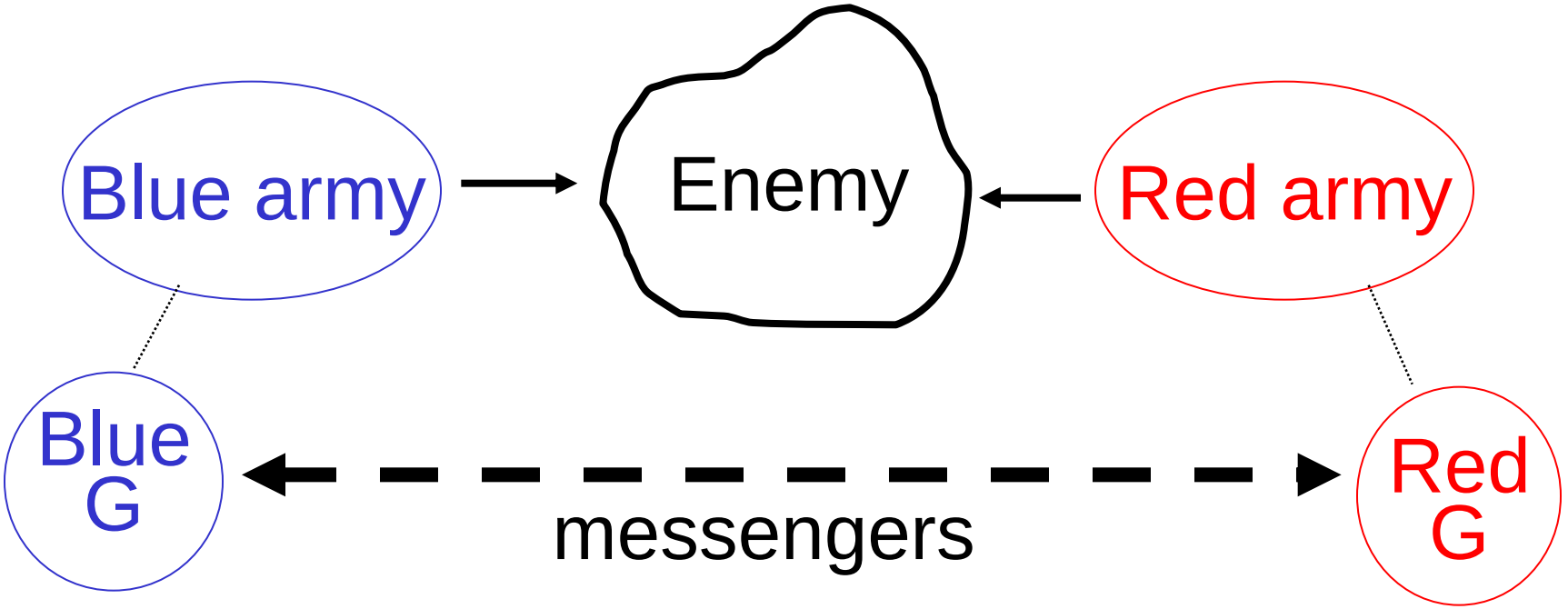
- Buď sa všetky uzly dohodnú na COMMIT transakcie, alebo sa všetky uzly dohodnú na ABORT transakcie
- Predpokladáme, že ak niektorý uzol spadne alebo sa zruší niektorá z liniek, tak ten uzol jednoducho prestane komunikovať (t.j. žiadne byzantínske chyby)
- Pozor, niektorý uzol môže spadnúť **počas vykonávania commit protokolu**

The one general problem (Trivial!)



➔ Battlefield

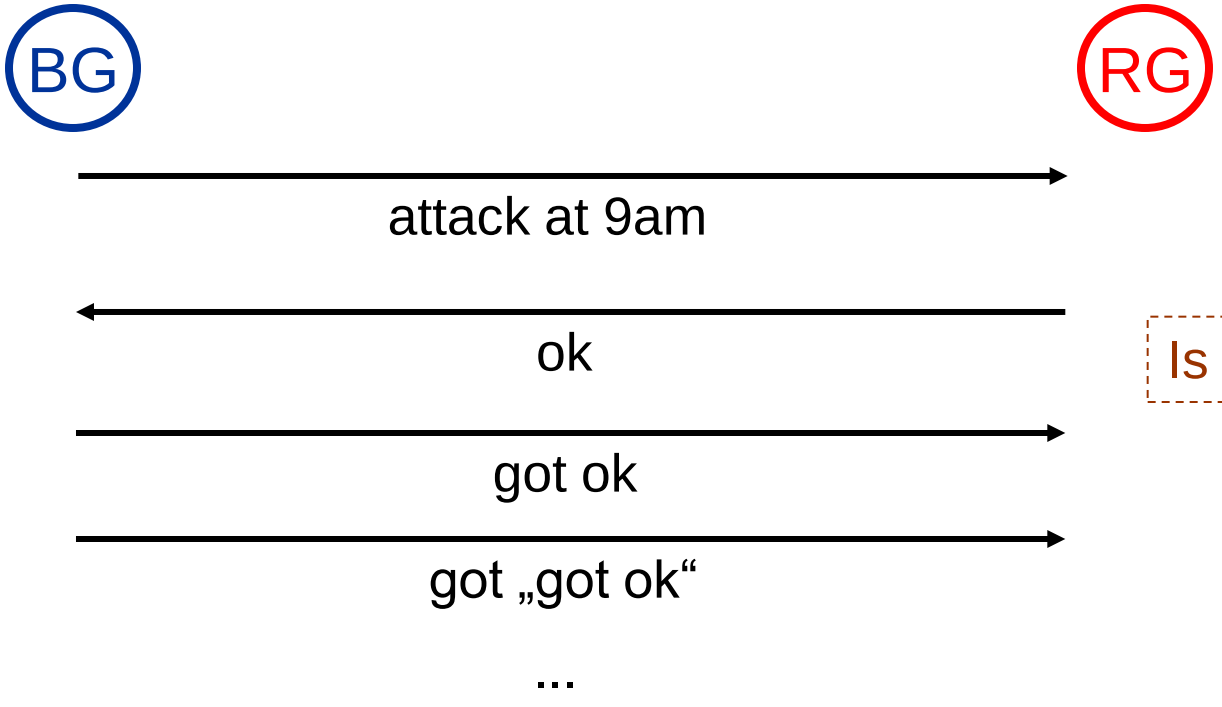
The two general problem:



- **Blue and red army must attack at same time**
- **Blue and red generals synchronize through messengers**
- **Messengers can be lost**

How Many Messages Do We Need?

assume blue starts...



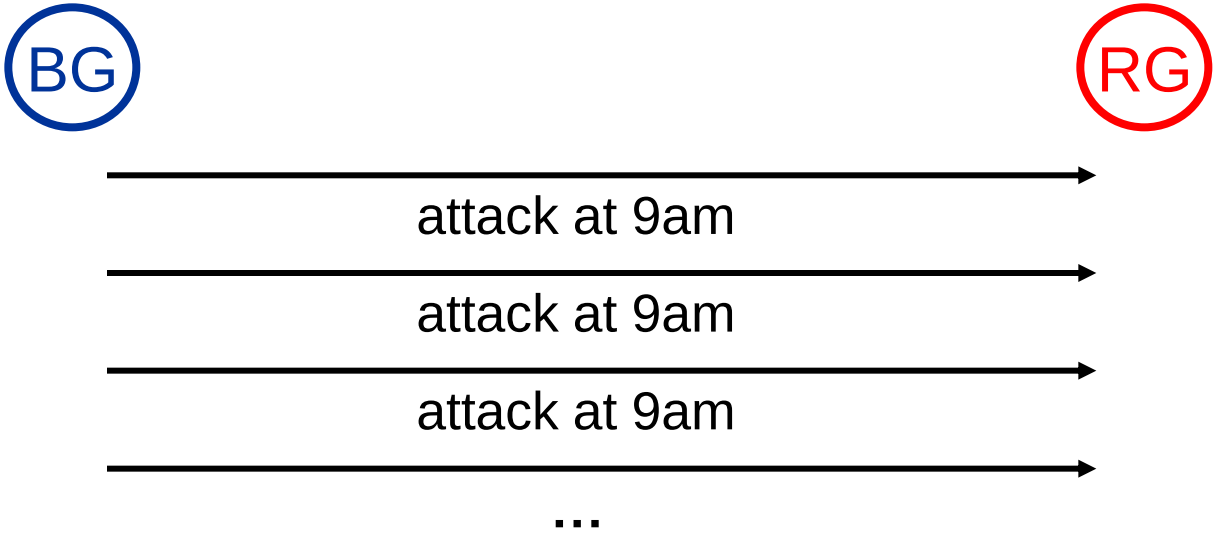
Is this enough??

There is no protocol that uses a finite number of messages that solves the two-generals problem (as stated here)

Probabilistic Approach?

- Send as many messages as possible, hope one gets through... Assume that eventually one does get through, so **keep sending until an answer arrives**

assume blue starts...



The red general does the same when he sends an answer. When the blue general receives an answer, he knows that his message (at least one messenger) got through!

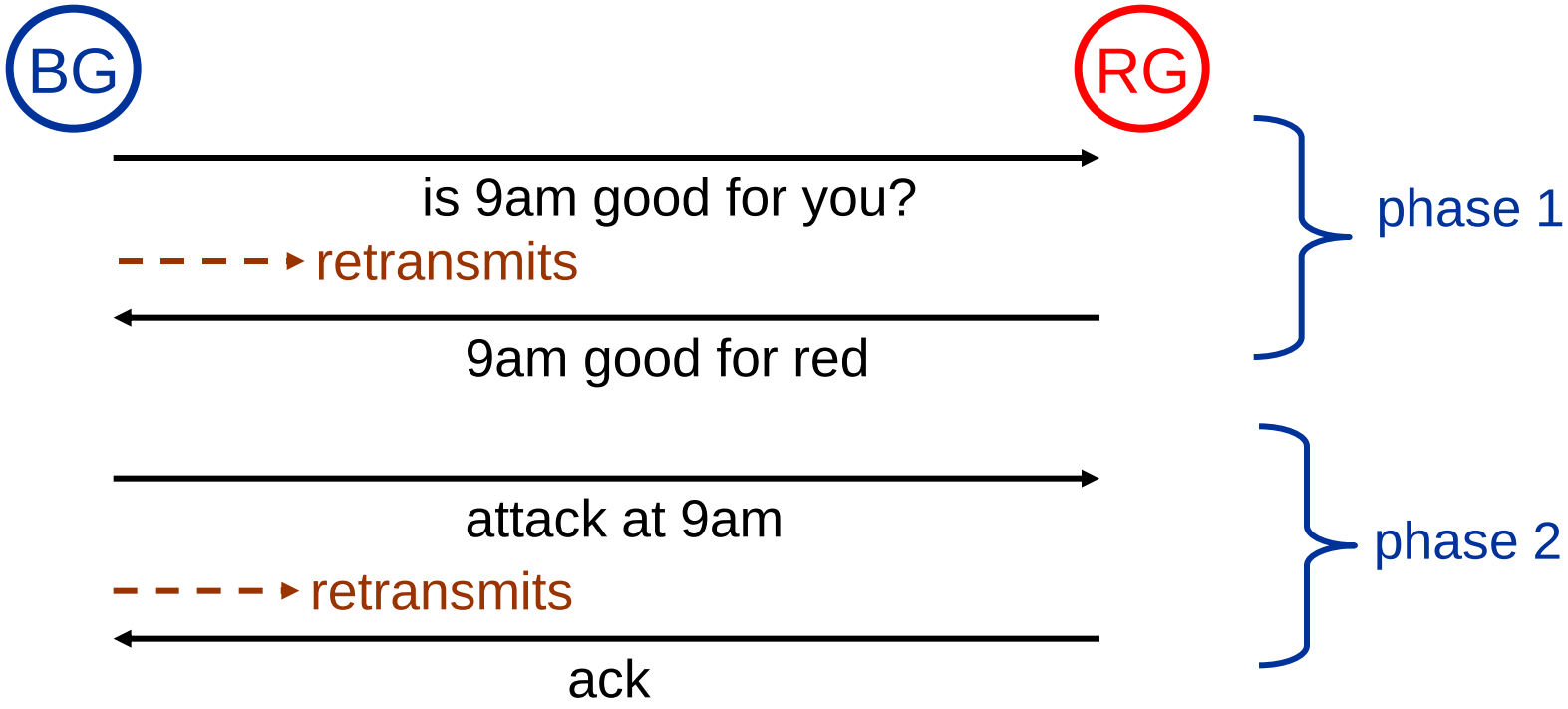
Probabilistic (?) Approach

- This is what indeed happens in contemporary networks (TCP/IP). Such a repetitive circulation of messages is hidden inside the TCP protocol. The circulation of messages (heart-beat) begins from the moment when a connection is established between two nodes, independently on whether the nodes actually attempt to communicate via that connection (ICMP packets)
- A good consequence is that from a point of view of a TCP application, the connection is reliable, i.e. messages do not get lost (and their ordering is preserved), if the connection works
- If the repeating trick stops working, then TCP gives up after a certain time has passed and breaks the connection. In such a case both the nodes are notified

Eventual Commit

- Eventually both sides attack...

assume blue starts...



This approach works for an arbitrary number of generals!

Atomicita: atomický commit protokol

Atomický commit, formálne požiadavky (Bernstein):

1. Všetky procesy, ktoré urobia rozhodnutie (Commit, resp. Abort), urobia rovnaké rozhodnutie
2. Keď sa proces rozhodne pre Commit, resp. Abort, tak svoje rozhodnutie už nezmení
3. Rozhodnutie Commit môže byť urobené len vtedy, ak všetky procesy hlasujú Yes (t.j. všetci s Commitom súhlasia)
4. Ak všetky procesy hlasujú Yes a nenastanú žiadne výpadky, tak rozhodnutie musí byť Commit
5. Ak po výpadku dostatočne dlho nie sú žiadne ďalšie výpadky, tak všetky procesy rozhodnutie urobia

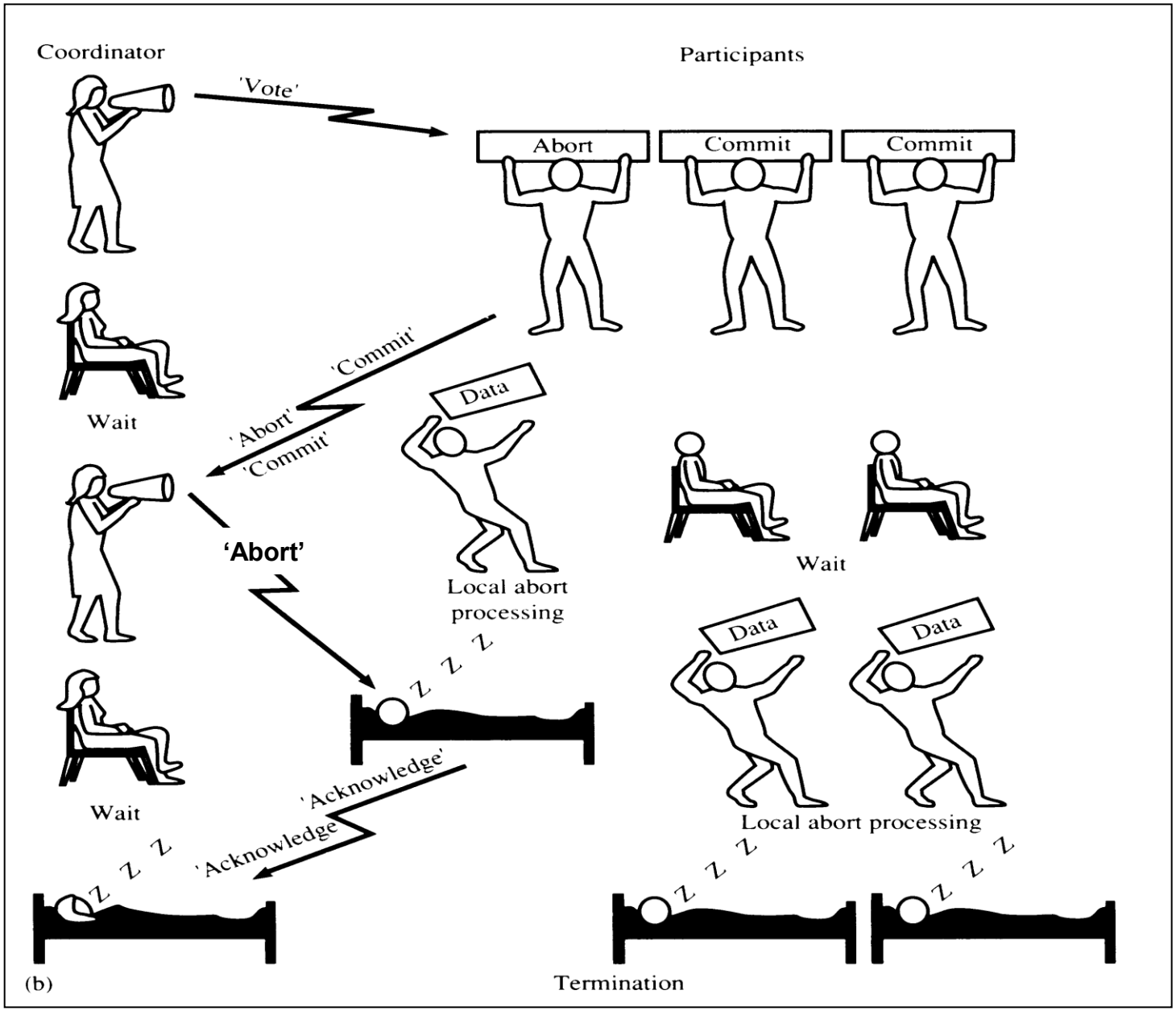
Atomicita: atomický commit protokol

• Atomický commit protokol začína uzol, ktorý od transakcie T dostal žiadosť o commit (t.j. domovský uzol transakcie T). Tento uzol funguje ako koordinátor. Ostatné uzly, ktoré sa týkajú transakcie T, fungujú ako participanti (koordinátor vie, kto sú participanti)

• Predpokladá sa asynchrónny model komunikácie. To znamená, že správy sa nestrácajú; a že medzi dvojicou procesov sú doručované v poradí, v ktorom boli odoslané. Neexistuje časová garancia pre doručenie správy, ale každá odoslaná správa raz doručená bude; výnimkou je prípad, ak adresát neexistuje (je spadnutý), vtedy sa správa zahodí. Ak uzol čaká na správu od iného procesu a ten iný proces spadne (alebo vypadne spojenie medzi nimi), tak čakajúci uzol ten výpadok detekuje

Atomicita: 2-fázový atomický commit protokol

Atomický 2-fázový commit protokol, ak žiaden uzol nespadne: ABORT



Atomicita: 2-fázový atomický commit protokol

2-fázový atomický commit protokol

• Ak niektorý uzol spadne a znovu sa obnoví, čo má robiť?

– **Nutnosť loggovania intencie (zamýšľanej akcie) pred vykonaním akcie, t.j. write-ahead log. Toto je v súlade s loggovaním v centralizovaných DBMS**

Atomicita: 2-fázový atomický commit protokol

2-fázový atomický commit protokol

1.fáza: transakcia T poslala koordinátorovi žiadosť o commit. Ak koordinátor sám rozhodne ABORT, tak sa táto fáza preskočí, koordinátor zapíše <ABORT T> do log-file a pošle správu [ABORT] všetkým participantom. Inak:

• Koordinátor zapíše do log-file <prepare T> a pošle správu [VOTE T] všetkým participantom

• Keď participant dostane správu o hlasovaní, zistí, či smie povoliť commit transakcie T:

–Ak nie (t.j. ak participant vie lokálne rozhodnúť ABORT), zapíše <ABORT T> a <NO T> do log-file a pošle správu [NO T] koordinátorovi

–Ak áno, zapíše <YES T> do log-file a pošle správu [YES T] koordinátorovi

Atomicita: 2-fázový atomický commit protokol

2-fázový atomický commit protokol

2.fáza: Koordinátor čaká na správy YES/NO od participantov

• Ak koordinátor dostane správu [NO T] od niektorého participanta, rozhodne ABORT, zapíše do log-file <ABORT T> a pošle participantom [ABORT T]

• Ak koordinátor dostane správu [YES T] od všetkých participantov, rozhodne COMMIT, zapíše <COMMIT T> do log-file a pošle participantom [COMMIT T]

• Keď participant dostane správu [COMMIT T] od koordinátora, zapíše do log-file <COMMIT T>, inak zapíše <ABORT T> (toto je rozhodnutie, ktoré sa už nikdy nezmení)

Atomicita: 2-fázový atomický commit protokol

2-fázový atomický commit protokol

Finálna fáza: Koordinátor čaká na potvrdenia od participantov

• Ak koordinátor dostane správu [ACK T] od všetkých participantov, môže na transakciu T zabudnúť

Táto fáza sa zdá byť zbytočná. Lenže inak musí koordinátor naveky držať záznam o commite transakcie T (lebo jeho správu <COMMIT T>, resp. <ABORT T> nemusel niektorý participant dostať)

Atomicita: 2-fázový atomický commit protokol

2-fázový atomický commit protokol

Ak spadne participant:

– Koordinátor zistí že participant spadol (detekuje timeout). V tom prípade rozhodne ABORT, zapíše <ABORT T> do log-file a pošle participantom správy [ABORT T]

– Keď sa spadnutý participant obnoví a nájde vo svojom log-file <COMMIT T>, urobí redo(T); ak nájde <ABORT T>, urobí undo(T); ak nájde
<NO T>, urobí undo(T); ak nájde [YES T], tak sa informuje o výsledku hlasovania u ostatných uzlov a až na základe toho urobí undo(T) resp. redo(T).

Atomicita: 2-fázový atomický commit protokol

2-fázový atomický commit protokol

Ak spadne koordinátor:

– Participant zistí že koordinátor spadol (detekuje timeout). Ak má tento participant v log-file <ABORT T> resp. <NO T>, tak sa vie rozhodnúť pre ABORT sám za seba; inak musí situáciu konzultovať s ostatnými participantmi

– Ak niektorý iný participant má v log-file <COMMIT T>, tak sa tento participant rozhodne pre COMMIT

– Ak niektorý iný participant má v log-file <ABORT T> resp. <NO T> (resp. nevie nič o hlasovaní, lebo spadol ešte skôr), tak sa tento participant rozhodne pre ABORT

– **Ak majú všetci participanti v log-file <YES T>, tak sa nevedia rozhodnúť a musia čakať na obnovu koordinátora (sú zablokovaní).** **Toto je neakceptovateľné!**

Atomicita: 2-fázový atomický commit protokol

Zaujímavý variant 2PC, decentralizovaný 2PC (len 2 kolá)

– koordinátor broadcastuje správu YES, resp. NO

– každý participant broadcastuje svoje YES, resp. NO

– po prijatí všetkých správ vie každý uzol lokálne rozhodnúť COMMIT, resp. ABORT

Atomicita: 2-fázový atomický commit protokol

Bernstein: ... “5. Ak po výpadku dostatočne dlho nie sú žiadne ďalšie výpadky, tak všetky procesy rozhodnutie urobia”

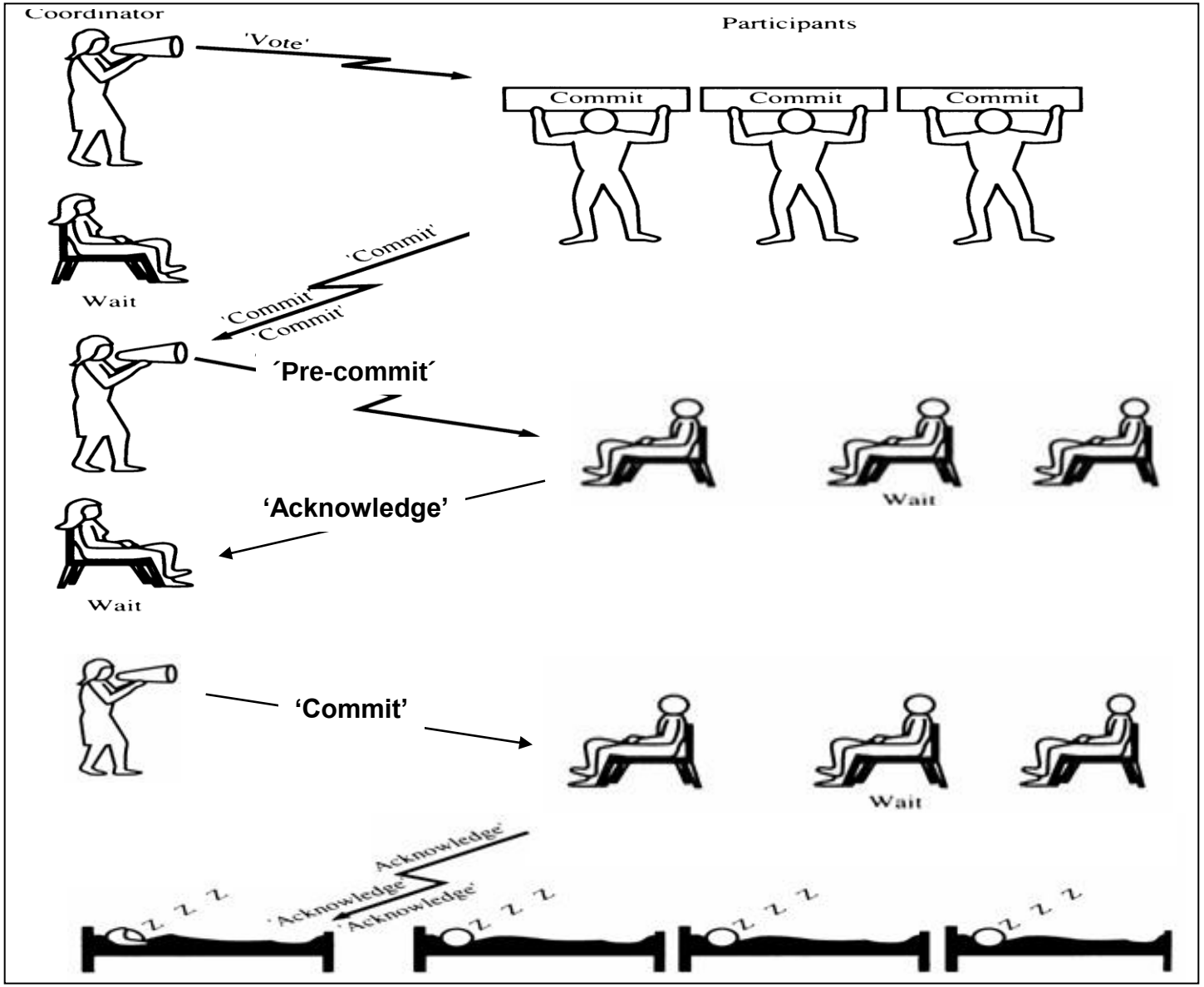
Je rozumné pridať ďalšiu požiadavku: 6. Všetky uzly, ktoré počas vykonávania atomic commit protokolu nespadnú, vedia urobiť rozhodnutie COMMIT resp. ABORT nezávisle od toho, kedy sa spadnuté uzly obnovia (t.j. aj keď sa žiaden zo spadnutých uzlov neobnoví)

2-fázový atomický commit protokol túto požiadavku nespĺňa. Ak v nevhodnom momente vypadne hoci len 1 uzol (koordinátor), tak žiaden z ostávajúcich uzlov nevie urobiť rozhodnutie skôr, ako sa koordinátor obnoví!

Neblokovací protokol existuje: stačí rozšíriť 2-fázový protokol o jednu fázu (fáza pre-commit) ⇒ 3-fázový protokol

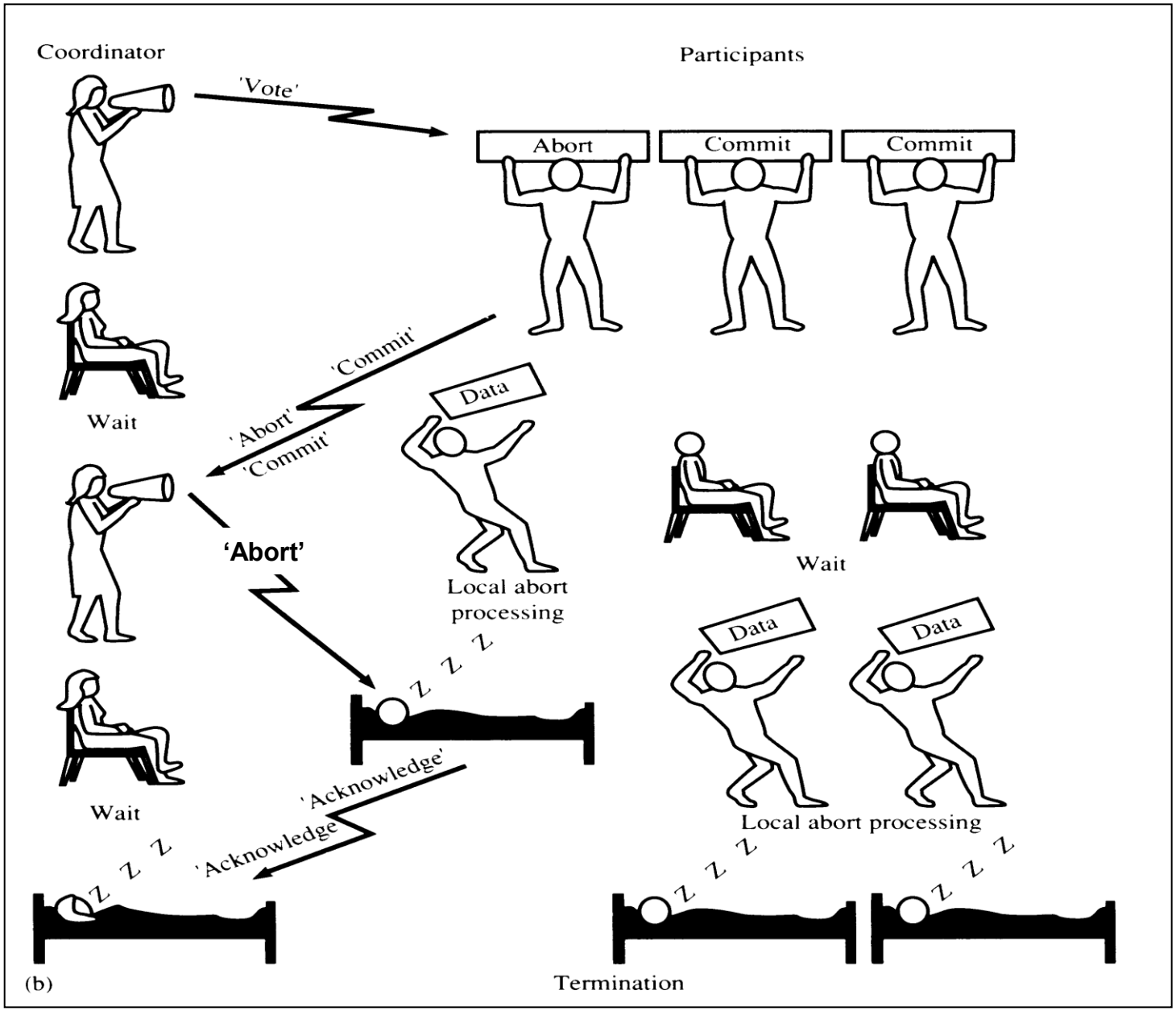
Distribuované transakcie: atomický commit

Atomický 3-fázový commit protokol, ak žiaden uzol nespadne: COMMIT



Distribuované transakcie: atomický commit

Atomický 3-fázový commit protokol, ak žiaden uzol nespadne: ABORT (tretia fáza je v tomto prípade zbytočná)



Distribuované transakcie: atomický commit

Idea dodatočnej (prostrednej) pre-commit fázy:

• ak koordinátor nespadne, tak 3-fázový protokol prebehne ako ten 2-fázový, len má o jednu „zbytočnú“ fázu navyše

• ak koordinátor spadne, tak transakcia môže byť commitovaná len vtedy, ak niektorý z nespadnutých uzlov dostal od koordinátora správu PRE-COMMIT (toto je rozdiel oproti 2-fázovému protokolu)

Ostáva vyriešiť otázku, ako zistiť, či niekto dostal od koordinátora správu PRE-COMMIT. Odpoveď: nespadnuté uzly si zvolia nového koordinátora. Ak má PRE-COMMIT nový koordinátor, nemusí sa nič pýtať; ak nie, opýta sa všetkých nespadnutých uzlov, stačí keď nájde jeden, ktorý PRE-COMMIT dostal

Ak nikto PRE-COMMIT nemá, nový koordinátor rozhodne ABORT; ak má, tak **najskôr** rozšíri PRE-COMMIT medzi všetky nespadnuté uzly a až potom rozhodne COMMIT

Distribuované transakcie: atomický commit

Ani 3-fázový protokol nerieši všetky problémy:

• Ak zlyhanie liniek rozdelí uzly do dvoch izolovaných podgrafov, tak izolované komponenty môžu urobiť rôzne rozhodnutia! Hrozí inkonzistencia!

Riešenie: Majoritný 3-fázový protokol

• treba pridať správu pre-abort [Bernstein ~1987]

• transakcia môže byť commitovaná, ak väčšina uzlov má pre-commit; môže byť abortovaná, ak väčšina uzlov má pre-abort; inak treba počkať na obnovu spadnutých uzlov (blokovanie)

• technický problém: po obnove treba niektoré uzly “konvertovať“ zo stavu commitable na stav abortable alebo naopak; dá sa to

• garantuje konzistenciu v prípade akýchkoľvek výpadkov

• za istých okolností sa zablokuje, ale menej často ako 2-fázový protokol [Skeen ~1982]

Distribuované transakcie: atomický commit

Oracle (a aj iné systémy) implementuje 2-fázový protokol, lebo je jednoduchší na implementáciu; a pritom sa spolieha na prozreteľnosť, ktorá snáď nenechá koordinátora zlyhať v nesprávnom momente príliš často